NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

APPLICATION OF A GENETIC ALGORITHM TO OPTIMIZE STAFFING LEVELS IN SOFTWARE DEVELOPMENT

by

Donald C. Johnson

December 1994

Thesis Advisor: Balasubramaniam Ramesh

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE APPLICATION OF A GENETIC ALGORITHM TO OPTIMIZE STAFFING LEVELS IN SOFTWARE DEVELOPMENT	5. FUNDING NUMBERS
6. AUTHOR(S) Johnson, Donald C.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Human resource management is gaining recognition as being one of the core attributes of effective software project management. With the great dependency the Department of Defense (DoD) has on systems software, the development of an optimal staffing policy that minimizes overall software project costs will be very valuable. This research aims at developing such a system using a dynamic simulation model that incorporates data collected during an actual software development project. A genetic algorithm is used to arrive at a solution to this nonlinear optimization problem. The simulation model accepts varying staffing schemes supplied from the genetic algorithm allowing for the examination of the effects of the staffing schemes on total project cost. The results indicate that the system was able to obtain a solution reducing overall project costs, but at the cost of additional workdays.

14. SUBJECT TERMS Human Resource Management, System Dynamics Model, Software Development, Genetic Algorithm, Staffing Policy, Simulation, Binary String, Optimization, Optimum Solution, Optimal Solution, Fitness Function, Generation.	15. NUMBER OF PAGES 73
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18

APPLICATION OF A GENETIC ALGORITHM TO OPTIMIZE STAFFING LEVELS IN SOFTWARE DEVELOPMENT

by

Donald C. Johnson
Lieutenant, United States Navy
B.T., University of Northern Iowa, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
December 1994

Author: Donald Craig Johnson
Donald C. Johnson

Approved by: [illegible]
Balasubramaniam Ramesh, Thesis Advisor

[illegible]
Tarek K. Abdel-Hamid, Associate Advisor

[illegible] for
David R. Whipple, Chairman
Department of Systems Management

ABSTRACT

Human resource management is gaining recognition as being one of the core attributes of effective software project management. With the great dependency the Department of Defense (DoD) has on systems software, the development of an optimal staffing policy that minimizes overall software project costs will be very valuable. This research aims at developing such a system using a dynamic simulation model that incorporates data collected during an actual software development project. A genetic algorithm is used to arrive at a solution to this nonlinear optimization problem. The simulation model accepts varying staffing schemes supplied from the genetic algorithm allowing for the examination of the effects of the staffing schemes on total project cost. The results indicate that the system is able to obtain a solution reducing overall project costs, but at the cost of additional workdays.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Billions of dollars in current and future Department of Defense (DoD) weapons and command, control, communications, and intelligence (C^3I) systems depend on high-performance, real-time computer systems that are capable of withstanding severe stresses without failing [Ref. 1:p. 1]. As DoD depends more and more on these systems to support its critical functions, the role of software becomes increasingly critical.

As technology advances, these DoD weapons and information systems capabilities are becoming increasingly dependent on complex software. This software, in turn, is rapidly becoming a very costly and technically challenging component. It has become apparent that overall systems costs are no longer driven by hardware, but by software [Ref. 2:p. 4]. It is estimated that DoD is annually spending between \$24 billion and \$32 billion on software costs alone. This accounts for between 8 and 11% of the entire DoD budget. By the year 2007, this number could soar to as high as \$50 billion, eventually accounting for up to 20% of DoD's overall budget. As a result, DoD software project managers are seeking better ways to effectively control software costs. The more quickly and efficiently high-quality software can be developed, the more flexible DoD will be in responding to changes and threats in an unstable world environment [Ref. 3:p. 10]. One way this can be done is through more effective management of project personnel. [Ref. 4:pp. 1-2]

B. HUMAN RESOURCE MANAGEMENT

Human resource management has gained recognition, in recent years, as being one of the core attributes of effective software project management. This is due to several reasons:

> Personnel costs are skyrocketing relative to computer hardware costs. Chronic problems in software development and implementation are more frequently traced to personnel shortcomings. Information systems staff sizes have mushroomed with little time for adequate selection and training. It is no wonder, then, that software managers find themselves focusing increasing amounts of attention on human resource issues [Ref. 5:p. 109].

If these personnel costs can be controlled, overall software project costs could be reduced.

One way these costs may be controlled is by minimizing personnel costs through optimization of the project staffing levels. This optimization process however, is difficult due to the complexities of the decisions involved. An effective project manager must first determine the size of the work force required to complete the project on schedule and then make any required adjustments if the project is either ahead or behind schedule. If new members are needed, they must be trained. The project manager must then take into account that during this training process, not only will the new project members be less than fully productive, but experienced project members who take time to train the new members will also be less than fully productive. It may take weeks just to build back up to the original productivity level at which the project was operating prior to hiring the new members. Eventually though, the productivity should surpass the original level. It is the project managers responsibility to weigh this information and determine if hiring new personnel would be beneficial. The determination of the staffing level can be thought of as a nonlinear optimization problem (minimizing the total project cost).

C. THE RESEARCH QUESTION

The primary research question that will be addressed in this thesis is how to compute an optimal software project staffing policy that will minimize total project costs. This research will utilize a dynamic simulation model of software development coupled with a genetic algorithm to investigate this question. The simulation model incorporates data from an actual software project and serves as a convenient and reliable alternative to experimenting with various staffing policies during an actual project. The genetic algorithm will be used as the optimization tool as it is considered to be suitable for nonlinear optimization problems such as this. Therefore, the modified research question is: Can a dynamic simulation model coupled with a genetic algorithm be used to compute an optimal software project staffing policy that will minimize total project costs?

D. METHODOLOGY

In this research, a genetic algorithm will be used in conjunction with a dynamic simulation model. The simulation model will accept varying staffing schemes supplied from the genetic algorithm, and while holding all other parameters constant, it will calculate the projected total development cost. Once the total cost is calculated, it is returned to the genetic algorithm to determine an "efficiency" value to help in the generation of potentially better schemes for further analysis. Through this optimization process, the combined system is theoretically able to determine a minimal solution. This approach is similar to that employed in [Ref. 6].

E. THESIS ORGANIZATION

This chapter introduces the need for minimizing software costs and the focus of the research. Chapter II is an introduction to the system utilized in the research and an explanation of its components. The chapter covers the genetic algorithm and how it works, the system dynamics model and its four subsystems, and the NASA DE-A Software Development Project. Chapter III is a detailed description of the system, its interfaces, and its operation. Chapter IV addresses the simulation runs that were conducted and gives an analysis of the results. The final chapter, Chapter V discusses the conclusions and recommendations for continued research.

II. SYSTEM COMPONENTS

A. THE GENETIC ALGORITHM

1. Introduction to the Genetic Algorithm

Genetic algorithms (GAs) are search algorithms that are conceptually based on the methods that living organisms use to adapt to their environment [Ref. 7:p. 7]. These methods, known as natural selection or evolution, combine the concept of survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search [Ref. 8:p. 1]. In each generation, a new set of string structures is created from (bits and pieces of) the fittest strings from the previous generation and occasionally a randomly altered new part. This process of exploiting historical data allows the GA to speculate on new search points that will improve performance thus producing better solutions.

Genetic algorithms were initially developed by John H. Holland, a professor of psychology and computer science at the University of Michigan. His research focused on what he called complex adaptive systems. These systems had the ability to overcome difficulties in their natural environment by reorganizing their component parts and adapting to their surroundings. Holland's goals were (1) to study and realize the adaptive processes of these natural systems, and (2) to mathematically model this evolutionary process and design artificial systems software that would utilize the mechanics of the natural systems. By the mid-1960s, Holland designed the first genetic algorithm which incorporated the theories of reproduction, mutation, and natural selection. [Ref. 7:pp. 8-9]

Since their development, genetic algorithms have been used as optimization tools for complex problems that involve numerous variables or involve combinations of linear and non-linear equations. As an optimization tool, the genetic algorithm attempts to improve performance leading to an optimal solution. In this process, there are two distinct steps, (1) the process of improvement and (2) reaching the optimum itself. Of these two steps, the most important is the process of improvement. In complex systems, due to the potential high costs involved, reaching the optimum (the one best) solution may not be justified as long as continuous improvement is being made and an optimal (desirable) solution can be found. [It would be nice to be able to reach the perfect solution, but with the increasing complexities of systems being developed today, we can only strive to reach the goal of improvement]. [Ref. 8:pp. 6-7]

2. How the Genetic Algorithm Works

When developing the original genetic algorithm, Holland selected the chromosome as the basic building block for his mathematical model. In his model, Holland replaced the chromosome (a string of genetic material) with a string of binary digits (ones and zeros). Each of the binary digits was then tied to the triggering of a specific action in a computer program that the string was designed to represent, much like bits of genetic materials present in chromosomes trigger certain occurrences or characteristics in the living organisms they represent. [Ref. 7:pp. 9-10]

Holland found that given a long enough string, any object could be characterized by the right combination of digits. He also found that any computer program capable of performing the combination of actions needed to arrive at the desired solution could also be represented in such a manner. To represent these computer programs, the binary string could range from several hundred to several thousand digits long [Ref. 7:p. 10]. With strings of this size, the time involved in trying every possible combination of digits to find the best solution would be endless. For a string 10,000 digits long (a conservative estimate for complex systems), there would be 2^{10000}, or approximately 10^{3000} possible combinations. To estimate how long it would take to search such a set, if 10^{12} strings could be tested each second, it would take longer than the age of the known universe just to search through 10^{100} combinations. [Ref. 7:p. 11]

Today, due to their unique approach, genetic algorithms are being employed to search the more complex systems. Genetic algorithms differ from the more traditional optimization and search procedures in four ways: (1) GAs use coding of the parameter set instead of the parameter set itself. (2) GAs search from a population of points instead of a single point. (3) GAs use objective function information instead of derivatives or other auxiliary knowledge. (4) GAs use probabilistic transition rules (randomized operators) instead of deterministic rules [Ref. 8:p. 7]. In addition to these unique procedures, genetic algorithms differ in the way their processes model the principles of evolution. It is through the processes of reproduction, crossover, and mutation that the GAs are able to arrive at an optimal solution [Ref. 6:p. 9].

a. Reproduction

The mechanics of a genetic algorithm are relatively easy to understand, they involve nothing more complex than creating character strings and swapping partial strings. To help illustrate the mechanics of a simple genetic algorithm, an example will be used [Ref. 8:pp. 7-17]. This example will consider the black box switching problem illustrated in Fig. 2-1 [Ref. 8:p. 8]. As pictured, the black box device consists of a bank of five input switches, each switch having an on and off position. For every setting of the five switches, there is an output signal *f*, mathematically $f = f(s)$, where *s* is a particular setting of the five switches. The objective of this problem is to determine the combination of switch settings that will produce the maximum *f* value.

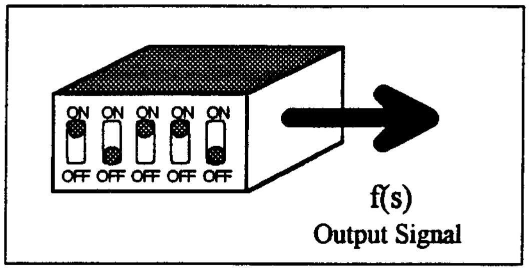


Figure 2-1: Black Box

The first step in a GA is to randomly generate a population of binary strings all of a predetermined length. In this example, a simple coding scheme can be devised by considering a string of five binary digits where each of the switch positions is represented by a 1 if the switch is on and a 0 if the switch is off. With this coding scheme, the string 10110 codes the switch positions pictured in Fig. 2-1. In this example, the random population was generated by using successive flips of an unbiased coin where a head = 1 and a tail = 0. This process might generate an initial population consisting of four strings ($n = 4$) as shown in Table 2-1.

TABLE 2-1

RANDOMLY GENERATED STRINGS

No.	String
1	01101
2	11000
3	01000
4	10011

Once the initial population is determined, the next step is to take this population and generate successive populations that will hopefully improve over time. The first operation in this process is reproduction. Reproduction is the process of copying of individual strings according to their objective function values, f (biologists and GA literature refer to this as the fitness function). Normally, the function f is some measure of profit or cost that requires optimization (either minimizing or maximizing depending on the case). Copying strings according to their fitness function means that the strings with the higher values have a greater probability of contributing one or more offspring to the next generation. This operator, an artificial version of natural selection, is the final judge of the string's existence.

The reproduction operator can be implemented in genetic algorithms in several ways. One way is to create a biased roulette wheel where each string in the current generation has a roulette wheel slot sized in proportion to its fitness function value. To determine a fitness value in this example, a binary string is converted to its equivalent base

10 number and then this new value (x_{10}) is squared. Table 2-2 lists the binary strings from the current generation, the corresponding base 10 numbers, and the calculated fitness function values.

TABLE 2-2

CURRENT GENERATION STRINGS AND FITNESS VALUES

No.	String	x_{10}	Fitness (x_{10}^2)	% of Total
1	01101	13	169	14.4
2	11000	24	576	49.2
3	01000	8	64	5.5
4	10011	19	361	30.9
Total			1,170	100

To determine the weighted roulette wheel slot sizes, the first step is to calculate the total fitness value over the entire population, in this case, the four strings. As shown in Table 2-2, this value is 1,170. The next step is to compute what percentage of the total value is contributed by each string (see Table 2-2). The example's corresponding biased roulette wheel for this generation's reproduction is pictured in Fig. 2-2.

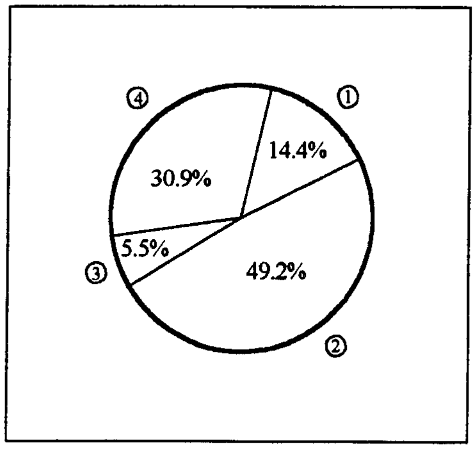


Figure 2-2: Biased Roulette Wheel

The final step in the process of reproduction is to create a new population of strings. This is done by spinning the roulette wheel n times, where n is the population size (recall that for this example $n = 4$). If the roulette wheel pictured in Fig. 2-2 was to be spun a total of four times, although the selection of the strings is probabilistic, it could reasonably be expected that string 1 and string 4 would each be selected once and string 2 would be selected twice. These strings are then entered into a tentative new population for further genetic operator action. The tentative new population, commonly referred to as the mating pool, consists of the strings listed in Table 2-3.

TABLE 2-3

TENTATIVE NEW POPULATION

No.	String	x_{10}	Fitness (x_{10}^2)
1	01101	13	169
2	11000	24	576
3	11000	24	576
4	10011	19	361
Total			1,682

It can be observed that as a consequence of reproduction, the overall fitness of the population has increased and the string with the poorest fitness value has momentarily disappeared. What has not occurred is the creation of new strings. If reproduction was the only genetic operator and the optimal solution had not been generated in the initial random selection of strings, the optimal solution could never be attained. Therefore, to arrive at the optimal solution, new strings must be created. To create these new strings, genetic algorithms employ a crossover operator [Ref. 6:p. 13].

b. Crossover

Simple crossover proceeds in two steps. In the first step, members from the mating pool are randomly grouped into pairs. In the second step, a number (k) is chosen randomly between one and the string length minus one [1, l - 1]. Two new strings are then created by swapping all characters after k. In the example problem, it will be assumed that string 1 was paired with string 2 with $k = 2$, and string 3 was paired with string 4 with $k = 3$. Before the crossover procedure, the strings would appear as shown in Fig. 2-3 (a) with crossover points indicated by the |. The four strings created during the crossover, indicated by the prime symbol ('), are shown in Fig. 2-3 (b). In examining the results of the crossover procedure, it is clear to see that three entirely new strings have been created. It is through this creation of new strings that the GA is able to search for a solution [Ref. 6:p. 15].

String 1 = 01|101   String 2 = 11|000

String 3 = 110|00   String 4 = 100|11

(a) Strings before crossover

String 1' = 01000 (Bits 1-2 from string 1, Bits 3-5 from string 2)

String 2' = 11101 (Bits 1-2 from string 2, Bits 3-5 from string 1)

String 3' = 11011 (Bits 1-3 from string 3, Bits 4-5 from string 4)

String 4' = 10000 (Bits 1-3 from string 4, Bits 4-5 from string 3)

(b) Strings after crossover

Figure 2-3: Simple Crossover

c. *Mutation*

To complete the creation of the next generation, the genetic algorithms employs a mutation operator. Mutation is the process where an element in a string is randomly changed (from a one to a zero, or vice versa) and it affects only a very small percentage of the population. In most cases mutations create strings that are less fit than the rest of the population. This occurrence exists in genetic algorithms to ensure that the population does not become too uniform. If this were to happen, the population would be incapable of further evolution and an optimal solution could not be reached. [Ref. 7:p.17]

To complete the generation process in the example, it will be assumed that the probability of mutation in this problem was 0.001 (one mutation per thousand bit transfers). With 20 bits transferred, it would be expected that only 20 x 0.001 or 0.02 bits will undergo mutation per generation. Therefore, no bit positions were changed during this generation due to mutation.

Having completed the reproduction, crossover, and mutation operators, the new generation can now be tested to see if there has been any improvement made on the initial generation. This is done by decoding the new binary strings and calculating their fitness values. The results from this process are listed in Table 2-4. By comparing the values in Table 2-2 and Table 2-4, it can be noted that the population's average fitness value (total fitness value divided by 4) has improved from 293 to 473 in only one generation. In the same period, the maximum fitness value has increased from 576 to 841. From this first generation, it is clear to see that the example is fulfilling the primary goal of genetic algorithms, to continually make improvement while searching for the optimal solution. Although random processes have helped cause these happy circumstances, according to Goldberg [Ref. 8], this improvement is by no means a fluke [Ref. 8: p 17].

TABLE 2-4

NEXT GENERATION STRINGS AND FITNESS VALUES

No.	String	x_{10}	Fitness (x_{10}^2)	% of Total
1	01000	8	64	3.4
2	11101	29	841	44.5
3	11011	27	729	38.6
4	10000	16	256	13.5
Total			1,890	100

3. The GAUCSD 1.4

The genetic algorithm system selected for use in this research was the GAUCSD 1.4 (Generic Algorithm, University of California, San Diego Version 1.4). GAUCSD 1.4 was developed at UC San Diego by Nicol Schraudolph and it is based on GENESIS 4.5, a genetic algorithm package written at the Naval Research Laboratory by John J. Grefenstette. This system was made available to encourage the experimental use of genetic algorithms on realistic optimization problems with the intent of helping identify the possible strengths and weaknesses of genetic algorithms. [Ref. 9:p. 1]

The GAUCSD 1.4 was selected based on several reasons. (1) Though most GAs being used today are custom-designed systems operated on workstations, there are several versions of GA software available for use on personal computers [Ref. 7:p. 21]. GAUCSD 1.4 is one of these versions. Although it was specifically written in the C programming language to operate under the UNIX operating system, it can easily be ported to work in the DOS environment used on most personal computers. (2) GAUCSD 1.4 is specifically designed to function as a minimization algorithm, the optimization technique that will be employed throughout the research. (3) It has well documented code in a ready to use format. Included in the documentation is an E-mail address which can be used any time there are any questions or feedback concerning the program. (4) In order to use the code for the research, it is only necessary to add a research specific fitness measure into the code. Any other modifications to the code will be unnecessary. (5) GAUCSD 1.4 has proven itself very reliable in previous research having similar system components as this project [Ref. 6].

B. SYSTEM DYNAMICS MODEL

1. A System Dynamics Model of Software Development

Simulation modeling provides an ideal opportunity to experiment with proposed solutions to highly complex problems. In addition to providing a more cost efficient and time saving approach, simulation allows control over the input parameters [Ref. 6:p. 20]. The software development process is one of the highly complex and dynamic processes that is adaptive to simulation. To simulate this process a comprehensive system dynamics model was developed for the purpose of studying, gaining insight into, and making predictions about the dynamics of the software development process [Ref. 5:p. 110].

The model was developed based on 27 focused field interviews with project managers from five separate software production organizations, supplemented by a large database of empirical findings created during extensive literature reviews. The model complements and builds upon current research efforts, which tend to focus on the micro components (e.g., scheduling, programming, testing, and productivity) by integrating the components of the software development process [Ref. 10: Preface]. This integrative model divides the software development and management activities into four main areas: (1) human resource management, (2) software production, (3) control, and (4) planning. Fig. 2-4 depicts the model's four main subsystems and some of the interrelationships that exist between them [Ref. 5:p. 110]. It is through the integration of these individual but interdependent components that the model is able to represent the total system.

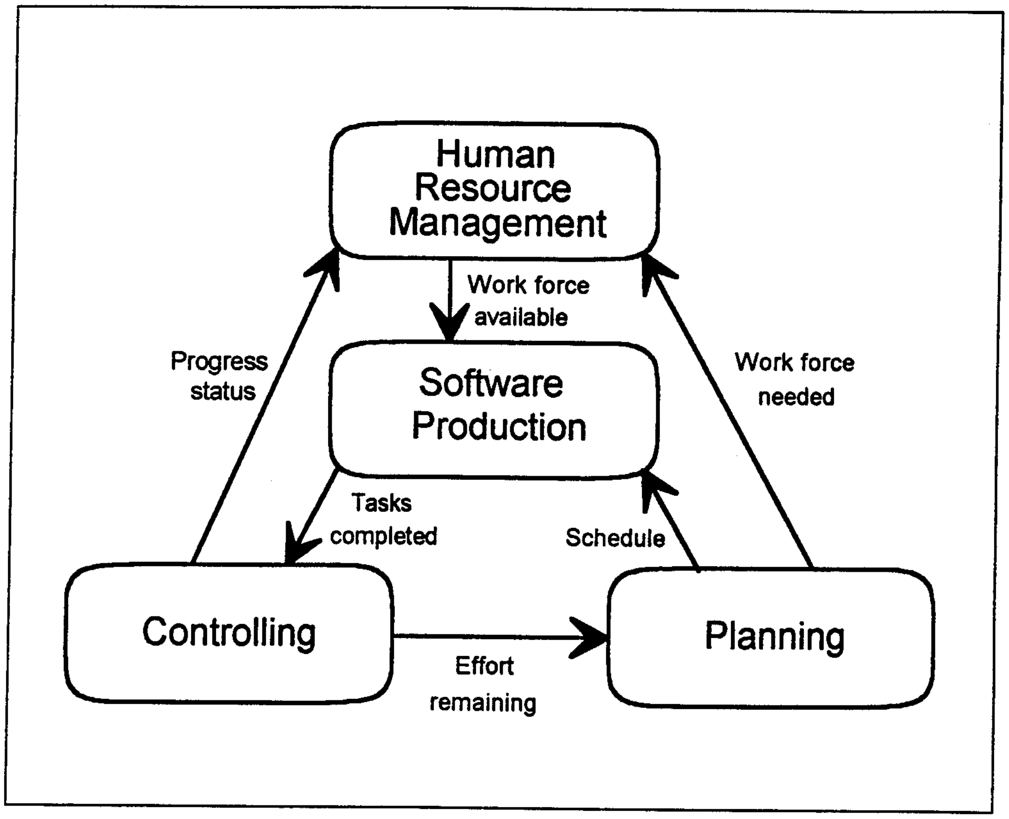


Figure 2-4: Overview of Model's Subsystems

Due to the complexity of the model, the following discussion is only intended as an introduction to the four subsystems, not an in-depth description. Interested readers can refer to Reference 10 for a more detailed description of the model and its subsystems.

a. The Human Resource Management Subsystem

The first subsystem to be introduced is the Human Resource Management Subsystem which includes the hiring, training, assimilation, and transferring of human resources. As depicted in Fig. 2-5, this subsystem divides a project's total work force into two main levels: (1) "Newly Hired Work Force" and (2) "Experienced Work Force." The first of two reasons for this division is due to the productivity differential between the two groups. Newly hired project members, which may include transfers from other parts of the organization, are less productive due to an orientation process that they must go through. The second reason for the division is to capture the training overhead involved when adding new members to a project. This is due to experienced workers taking time away from their other work to help train the new members, thus reducing their own productivity on an average of 20%. [Ref. 10:pp. 63-65]

Some of the significant decisions that are generated within the human resource management subsystem include: (1) determining an appropriate level for the "Total Work Force," (2) determining a hiring goal/strategy for the software project, (3) determining the "Work Force Level Sought," and (4) estimating an annual turnover rate. It should be noted that these activities are not conducted purely within the human resource management subsystem, instead, they take various inputs from the other main subsystems.

These interdependencies in turn help support and strengthen the integration throughout the entire dynamics model. [Ref. 10:pp. 65-68]

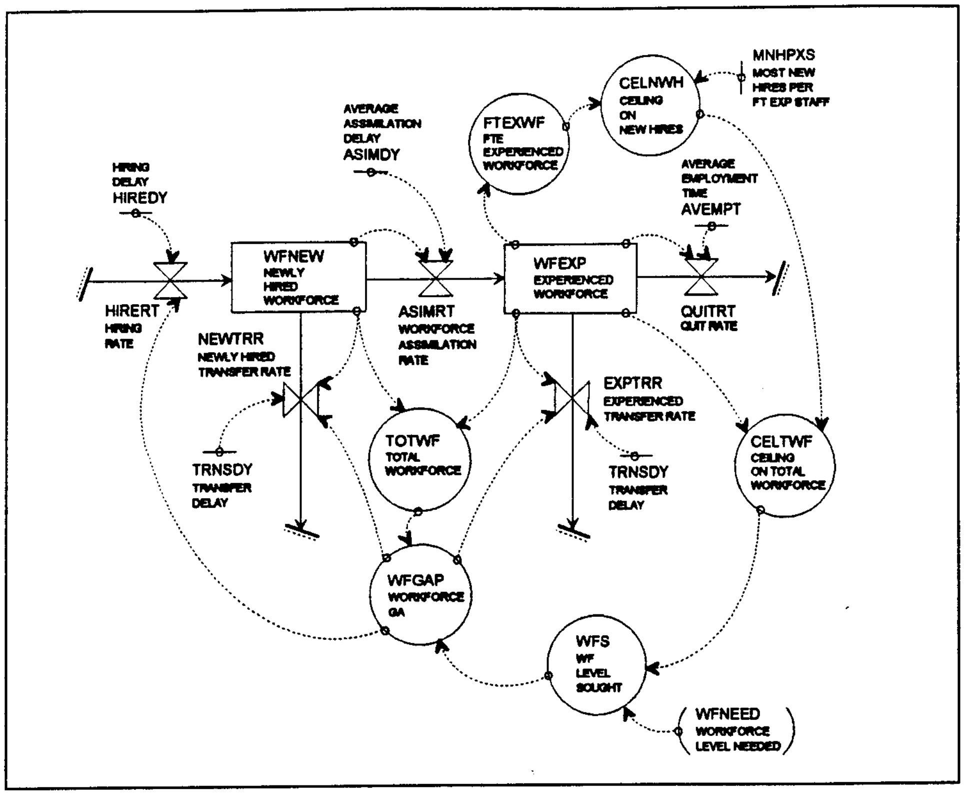


Figure 2-5: The Human Resource Management Subsystem

b. The Planning Subsystem

The second subsystem in the model is the Planning Subsystem. In the Planning Subsystem, depicted in Fig. 2-6, initial project estimates are made at the start of the project and revised as required during the project's life. The project estimates that are calculated within this subsystem include: (1) staffing load, (2) completion time, and (3) man-days remaining. For example, if a project is perceived to be behind schedule, organizational plans can be revised to add more people to the project, extend the scheduled completion date, or do a combination of the two. However, it should be noted that staffing decisions are not determined based solely on scheduling considerations. Consideration is also given to factors determined in the Human Resource Management Subsystem (i.e., training requirements and stability of work force). By integrating outputs from the two subsystems, the model is able to determine the appropriate project staffing levels. [Ref. 5:pp. 115-116]

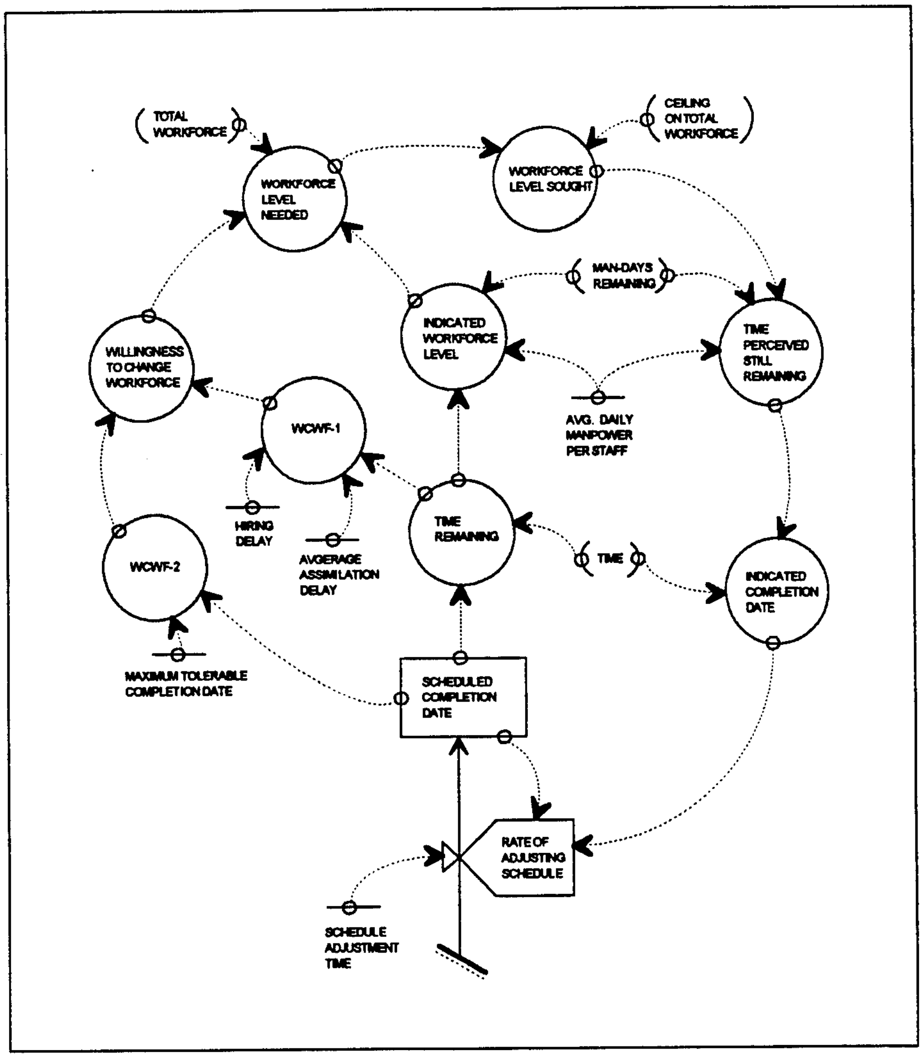

TOTAL WORKFORCE
WORKFORCE LEVEL NEEDED
WORKFORCE LEVEL SOUGHT
CEILING ON TOTAL WORKFORCE
MAN-DAYS REMAINING
INDICATED WORKFORCE LEVEL
TIME PERCEIVED STILL REMAINING
WILLINGNESS TO CHANGE WORKFORCE
WCWF-1
AVG. DAILY MANPOWER PER STAFF
HIRING DELAY
AVGERAGE ASSIMILATION DELAY
TIME REMAINING
TIME
WCWF-2
INDICATED COMPLETION DATE
MAXIMUM TOLERABLE COMPLETION DATE
SCHEDULED COMPLETION DATE
RATE OF ADJUSTING SCHEDULE
SCHEDULE ADJUSTMENT TIME

Figure 2-6: The Planning Subsystem

c. *The Software Production Subsystem*

The third subsystem in the model is the Software Production Subsystem. This subsystem includes the development, quality assurance, rework, and system testing aspects of the project's software. The normal progression through these four activities commences in the development phase where the software is designed and then coded. Concurrent with the software development, the code is reviewed using various Quality Assurance (QA) techniques to locate any design/coding errors. Errors detected through these activities are then reworked. Any errors that remain undetected during the QA checks will normally be caught and corrected during the testing phase, but this will be at a high cost. It is QA's goal to detect and correct as many errors as early as possible in the development process to keep rework costs low. If errors are allowed to remain undetected until the testing phase, their effects can escalate causing the rework costs to increase significantly over what they would have been if the error were found during any of the earlier stages. [Ref. 10:p. 69]

d. *The Controlling Subsystem*

The final subsystem in the model is the Controlling Subsystem. It consists of three elements: (1) measurement, (2) evaluation, and (3) communication. Measurement is defined as the detection of what is happening in the activity being controlled, evaluation is the assessment of its significance, and communication is reporting what has been measured and assessed so the behavior can be altered if need be. To relate this to software production, progress can be measured by the number of resources consumed, tasks completed, or both. Once an assessment has been made that the project is behind or ahead of schedule, the project can be altered to adjust to the situation. [Ref. 10:p. 117]

An ever challenging task in controlling the software production is determining the project's overall progress. Since software is an intangible product for the majority of the development process, it is often difficult to determine the projects progress during intermediate phases. The purpose of this subsystem is to accurately track the development progress during all stages of the process and adjust schedules or manpower as required to achieve anticipated results. [Ref. 10:p. 119]

2. Project Staffing

Determining the staffing levels needed in a software project is a challenging and vital task. This task, however, can be made easier with the assistance of the system dynamics model that has been described in this chapter. In the model, the staffing decisions are determined based on a joint effort between the planning and human resource management subsystems. When making the determination, the subsystems look at factors such as scheduling concerns, training requirements, and stability of work force.

By dividing the value of "Man-Days Remaining" by the "Time Remaining" the Planning Subsystem is able to determine the "Indicated Work Force Level" (i.e., the work force size believed to be required to finish the software project on the currently scheduled completion date). Once calculated, if the "Indicated Work Force Level" is less than the "Total Work Force" (i.e., current number of full-time-equivalent employees), excessive employees could be transferred from the project. If the "Indicated Work Force Level" is greater than the "Total Work Force," then management could decide to hire new employees or transfer-in employees currently working on other projects. [Ref. 10:p. 129]

Before the decision to hire new employees can be made, the Human Resource Management Subsystem must consider the training requirements for the project and the stability of the work force. That is, before hiring or transferring any new members into a project, management must estimate how long the new members will be needed. Each organization weighs this factor differently, but in general, the relative weighing between the desire to have a stable work force and the desire to complete the project on time changes dynamically throughout the life of the project. For example, toward the end of the project, some managers are reluctant to hire new people, even if the project is running behind schedule. These managers feel that it would take too much time and effort (relative to the time and effort remaining) to hire new people, acquaint them with the mechanics of the project, integrate them into the project team, and train them in the technical areas of the project. [Ref. 5:p. 112]

a. Adjusting the Work Force Level

The decision whether to hire new employees or not is made operational in the system dynamics model by means of the following equation:

$$\begin{aligned} \textit{Work Force Level Needed} = {} & \textit{Indicated Work Force Level} \times \textit{WCWF} + \\ & \textit{Total Work Force Level} \times (1 - \textit{WCWF}). \end{aligned}$$

As shown, the "Work Force Level Needed" is the weighted average between the "Total Work Force Level" and the "Indicated Work Force Level." That is, it accounts for both the stable work force level and the number of workers required to finish the project on schedule. This formulation applies only when the "Indicated Work Force Level" is greater than the "Total Work Force Level," an indication that a larger project work force is needed. When the opposite condition exists, the "Work Force Level Needed" would simply be set to the lower value, and excess employees would be transferred out of the project. [Ref. 10:pp. 130-131]

The weighting factor (WCWF) in the equation above is termed the "Willingness to Change Work Force." It is a variable that can assume values from 0 to 1, inclusive. When WCWF = 1, the weighting considers only the "Indicated Work Force Level," i.e., management would adjust the work force size to a level it feels is required to complete the project on schedule, possibly compromising the work force stability. As the WCWF variable moves towards 0, work force stability comes more and more into play. Finally, when WCWF = 0, the weighting is based entirely on work force stability, i.e., management will maintain the project's work force at the current level. [Ref. 5:p. 112], [Ref. 10:p. 131]

The "Willingness to Change Work Force" variable consists of two components, namely, WCWF-1 and WCWF-2 (refer back to Fig. 2-6). The first component, WCWF-1, represents the management pressures that develop over work force stability as the project progresses into its final stages. An example WCWF-1 policy curve is pictured in Fig. 2-7. To understand what the figure depicts, one must first assume that the "Willingness to Change Work Force" consists only of WCWF-1. In the early phases of the project when "Time Remaining" is ordinarily much larger than the sum of the "Hiring Delay" plus the "Average Assimilation Delay," WCWF would be equal to 1. Here management would be willing to adjust the work force to whatever level it feels is necessary to complete the project on schedule. As illustrated in the figure, when the "Time Remaining" falls below 1.5 x (Hiring Delay + Average Assimilation Delay), the reluctance to hire new people begins to build. Later, when "Time Remaining" drops below 0.3 x (Hiring Delay + Average Assimilation Delay), there will be no more additions to the work force and the hiring rate will fall to zero. From this point on, if the project falls behind schedule, management could only react by pushing back the scheduled completion date.

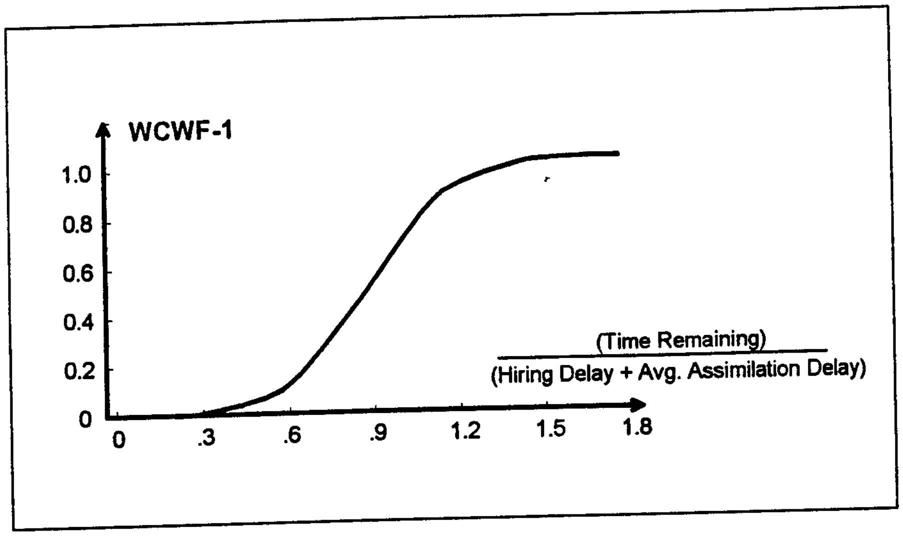


Figure 2-7: General Form of WCWF-1

b. *Schedule Stability*

When software development is part of the critical path of overall system development, as in projects involving embedded software for weapons systems, serious software schedule slippages must be avoided. If not, slippages in the software schedule may result in very costly slippages in the delivery schedule of the overall system. In projects such as these, as long as the "Scheduled Completion Date" is comfortably lower than the "Maximum Tolerable Completion Date," the decisions to add more people, adjust the schedule, or do a combination of the two will continue to be based on the "Willingness to Change Work Force" policy represented by WCWF-1. However, as the "Scheduled Completion Date" approaches the "Maximum Tolerable Completion Date," pressures start to develop that override the work force stability concerns. In such software projects, management often hires more people to avoid overshooting the "Maximum Tolerable Completion Date." [Ref. 10:pp. 132-133]

The progression of the overriding schedule pressures are represented by the "Willingness to Change Work Force" function;

$$WCWF = MAXIMUM\,(WCWF\text{-}1,\ WCWF\text{-}2),$$

where WCWF-2 is the policy curve illustrated in Fig. 2-8. As shown in the figure, as long as the "Scheduled Completion Date" is comfortably lower than the "Maximum Tolerable Completion Date," the value of WCWF-2 equals zero and has no bearing on the determination of WCWF or the hiring decision (remember, at this time the WCWF policy is represented by WCWF-1). However, as the "Scheduled Completion Date" approaches the "Maximum Tolerable Completion Date," schedule pressures start to develop causing WCWF-2 to rise gradually. Because this situation develops near the completion of the project, the value of WCWF-1 is probably approaching or at zero. When the value of WCWF-2 exceeds WCWF-1, the "Willingness to Change Work Force" function becomes totally dominated by scheduling in the attempt to avoid overshooting the "Maximum Tolerable Completion Date." [Ref. 10:p. 133]

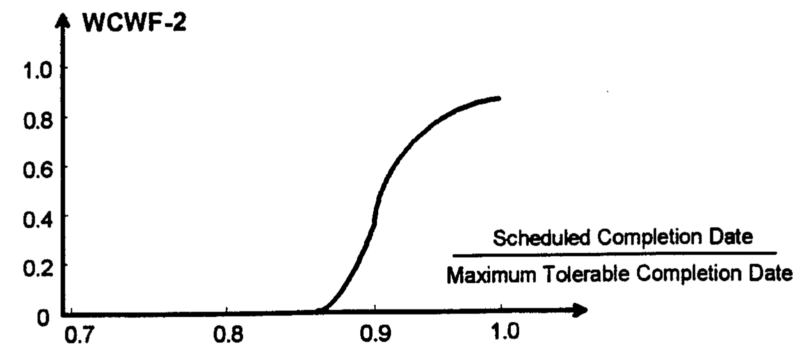


Figure 2-8: General Form of WCWF-2

Two final notes concerning the "Willingness to Change Work Force" function. First, when used to represent projects in which there are no rigid time commitments, the user only has to set the value of "Maximum Tolerable Completion Date" at some high value. This action will keep WCWF-2 equal to zero thus making WCWF solely a function of WCWF-1. Second, the WCWF variable represents a management *policy* for software projects. A range of policies is possible, representing strategies used by managers to balance work force and schedule adjustments in the attempt to minimize costs and schedule overruns throughout the life of the project. [Ref. 10:p. 134]

3. The DYNAMICA SD

The DYNAMICA SD (DYNAMICA Model of Software Development) is the system dynamics model chosen for use in this research project. It is a PC based program designed to simulate software project models. In this model, the staffing level is regulated by the

"Willingness to Change Work Force" (WCWF) variable which is determined dynamically throughout the project's development life cycle. Recall, this variable is determined by the function;

$$WCWF = MAXIMUM\,(WCWF\text{-}1,\ WCWF\text{-}2).$$

Once determined, these values are then entered into the model at the dynamically designated points. As illustrated in the function, when attempting to find an optimal staffing solution by varying the 11 WCWF values, thus introducing a variety of differing staffing policies, it is necessary to vary either or both of the corresponding WCWF-1 and WCWF-2 values.

C. THE NASA DE-A SOFTWARE DEVELOPMENT PROJECT

The DE-A software project was conducted by the Systems Development Section of NASA's Goddard Space Flight Center (GSFC) located in Greenbelt, MD. The primary requirements for the project were to design, implement, and test a ground-based software system capable of processing telemetry data in order to provide altitude determination and control for NASA's DE-A satellite. [Ref. 5:p. 118], [Ref. 10:p. 139]

The DE-A software project was selected as an in-depth case study in order to test the system dynamics model (i.e., examine the model's ability to reproduce the dynamic patterns of a completed software project). The project was selected because it satisfied three criteria set by the model's designers: it was (1) medium in size (i.e., 16-64 KDSI), (2) recent, and (3) "typical," (i.e., developed in a familiar in-house software development environment). [Ref. 10:p. 139]

The results from the DE-A case study will be used as the knowledge base in this research project. All results from the DE-A project are documented and all of the over 100 variables used in the model are known [Ref. 10]. By changing only the values of either or both of the WCWF-1 and WCWF-2 variables at each of the 11 intervals, it will be possible to evaluate the effect that the new WCWF policy has on the project outcome. This information will then be passed to the genetic algorithm to be processed in the attempt to determine the work force levels that will minimize project costs.

III. THE GASD SYSTEM

A. SYSTEM OVERVIEW

The system chosen for this research is referred to as the Genetic Algorithm Software Development System or GASD. GASD has previously been used to minimize software development costs by searching for optimal QA levels [Ref. 6]. This chapter primarily highlights the changes that were necessary to conduct the simulations for this research project.

The Genetic Algorithm Software Development System consists of the DYNAMICA simulation program, the GAUCSD program, and a fitness function compatible with DYNAMICA and the GAUCSD. The three components and their corresponding interactions are depicted in Fig. 3-1. As illustrated, the fitness function is the heart of the GASD system. Upon system initiation, the GA creates the WCWF-1 variables (through genetic

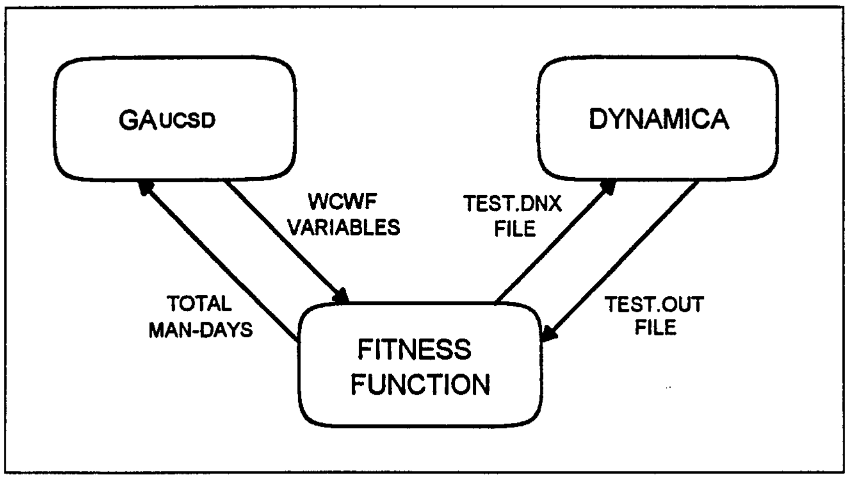


Figure 3-1: GASD System

operation) and sends them to the fitness function for evaluation. The fitness function passes this along to the DYNAMICA program through a file named TEST.DNX. The fitness function then executes the DYNAMICA program. During execution, the DYNAMICA program writes a TEST.OUT file, containing the total development efforts for the project. The fitness function then extracts the Total Man-Days value and returns it to the GA as the fitness of the GA string representing the WCWF-1 variable. This entire process, which comprises a single trial, is then repeated as necessary, depending on the length of the simulation run.

B. EXECUTING THE GAUCSD

There are four steps involved in preparing to use a conventional genetic algorithm like the GAUCSD used in this research. These steps include the determination of four factors: (1) the representation scheme, (2) the fitness measure, (3) the parameters and variables for controlling the genetic algorithm, and (4) how to display the results and the criterion for terminating a simulation run.

1. Determining the Representation Scheme

The first step in preparing to use the GAUCSD is to determine an applicable representation scheme for the variables involved. As discussed in Chapter II, genetic algorithms typically process only binary strings. Therefore, a scheme must be devised that allows the WCWF variables to be represented as a binary number or string.

As stated in an earlier discussion, the WCWF variable must be a real number between 0 and 1 inclusive. This range can be represented with seven binary digits since 2^7

allows for a range from 0 to 127. If these numbers where then divided by 100, the range of 0 to 1.27 would be possible. For those possible numbers greater than 1, a simple "if" statement will be used in the fitness function to enter the variable as a 1 (i.e., if *var* > 1, then *var* = 1).

Once the number of binary bits per variable is determined, the next step is to determine the binary string length. Since there are ten points in the simulation that require a WCWF variable (there are actually eleven points, but the last point must always be equal to 1), it will be necessary to have a binary string length equal to 10 (# of WCWF variables determined by GA) times 7 (# of binary bits per variable) for a binary string (chromosome) length of 70.

2. Determining the Fitness Measure

The next step in preparing to use the GA is to determine the measure of fitness that will be used. Recall, the objective of this research is to determine if it is possible to predict an optimal staffing policy that will be able to minimize total project costs. In order to do this optimization problem, a variable must be chosen that can reflect these costs. The Total Man-Days or Cumulative Man-Days (CUMMD) variable was chosen because it is the sum of all the man-days spent throughout the entire project and it directly reflects the total project costs (i.e., the less man-days spent, the lower the overall cost).

3. Determining the Parameters for Controlling the GA

The third step in preparing to use the GAUCSD is determining the parameters that will be necessary in controlling the GA. The GAUCSD is controlled by a SAMPLE.IN file that is input at the start of each run. An example of the SAMPLE.IN file is illustrated in Fig. 3-2. The parameters that are of primary concern include the total number of generations, the crossover rate, the structure length, and the mutation rate.

```
Experiments = 1
Total Trials = 90000
Population Size =900
Structure Length = 70
Crossover Rate = 0.600000
Mutation Rate = 0.005000
Generation Gap = 1.000000
Scaling Window = -1
Report Interval = 1000
Structures Saved = 10
Max Gens w/o Eval = 2
Dump Interval = 1
Dumps Saved = 1
Options = Aaeclu
Random Seed = 3436473682
Maximum Bias = 0.990000
Max Convergence = 30
Conv Threshold = 0.950000
DPE Time Constant = 40
Sigma Scaling = 2.000000
```

Figure 3-2: Example SAMPLE.IN File

In the GAUCSD program, the number of generations to be run is determined by dividing the Total Trials by the Population Size. As shown in the example file, the number of generations to be run would be equal to 90,000/900 or 100. It should be noted that with larger population sizes (greater than 500) as in this example, it may not be necessary to have a large number of generations. But, for smaller population sizes, to get the same accuracy, it may be necessary to increase the number of trials in order to increase the total number of generations.

The last three parameters that concern this project will be determined as follows: (1) The crossover rate is varied between 0.6 and 0.9 at intervals of 0.1. Crossover rates any higher (i.e., greater than 1.0) would result in crossing over at more than one location, a condition that is not necessary for short string lengths of 70. (2) The structure or string length for the GA runs will remain constant at 70. (3) The mutation rate will intentionally remain low as discussed in Chapter II. All other parameters in the SAMPLE.IN file are kept constant. Interested readers can refer to [Ref. 9:pp. 14-17] for a detailed description of the remaining parameters.

4. Determining When to Terminate a Run and Displaying the Results

The final step in preparing to use the GAUCSD is determining the criteria used for terminating a simulation run and deciding how to display the results. To terminate a run, the GAUCSD uses one of two methods. The first method ends the run when the total number of trials has been completed. The second method terminates the simulation when a predetermined convergence threshold has been reached. Both methods of ending a run are specified within the SAMPLE.IN file. If a simulation was run using the example file in Fig. 3-2, the run would terminate when the convergence threshold of 95% was reached or when 90,000 trials were completed, whichever occurs first.

To help explain how the convergence threshold method of termination works, the reader must understand the convergence process. A run converges when the genetic makeup of a population is not significantly different enough to produce a better result and the only way to introduce any variety into the population is through the process of mutation. This means that in the example depicted in Fig. 3-2, with the convergence threshold set at 95%, the simulation run would terminate when 95% of the binary strings in any generation converged.

Once the run has terminated, the data must be arranged in a usable format. This is done by examining the results that were saved by the genetic algorithm. During its execution, the GAUCSD saves data in several files. Four of these files and a description of how and when they are updated by the GA are depicted in Fig. 3-3.

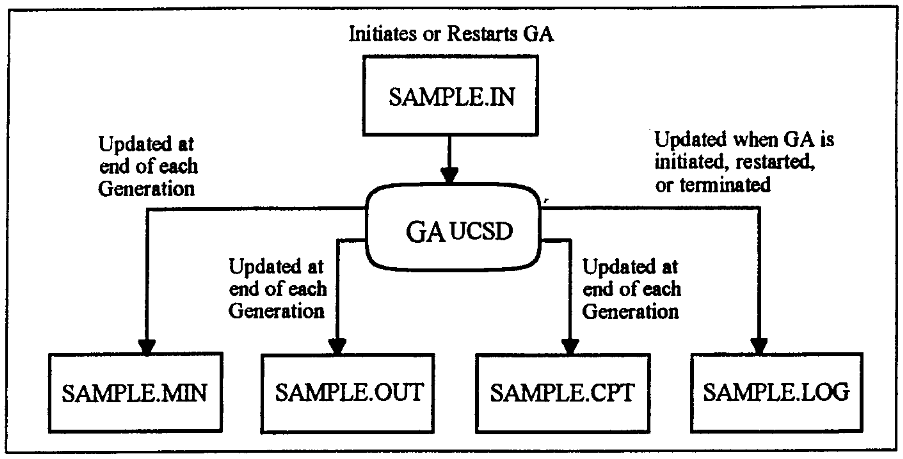

Initiates or Restarts GA
SAMPLE.IN
Updated at end of each Generation
Updated when GA is initiated, restarted, or terminated
GA UCSD
Updated at end of each Generation
Updated at end of each Generation
SAMPLE.MIN
SAMPLE.OUT
SAMPLE.CPT
SAMPLE.LOG

Figure 3-3: GAUCSD File Handling

The first of the files pictured in Fig. 3-3 is the SAMPLE.MIN file. This file is used to save the better results that are achieved during the GA's execution. The total number of results that are saved is dependent upon the Structures Saved parameter which is found within the SAMPLE.IN file. In the example file shown in Fig. 3-2, the input value is 10 (i.e., if the GA were to be run, the 10 best results would be saved). In addition to saving the better results, the SAMPLE.MIN file lists both a decimal number and its binary representation for each of the ten WCWF-1 variables used in the DYNAMICA program. It should be noted that the GAUCSD incorporates gray coding in the binary representation scheme, therefore, manual translation of the binary strings into the decimal numbers is impractical. Fig. 3-4 depicts an example SAMPLE.IN file.

```
0000010 1001010 1010100 1100111 0001000 0000010 0000101 1111101 1011001 1000011   1.8499e+03   53 39881
-0.6024 0.519469 0.39265 0.0587991 -0.483911 -0.609594 -0.575274 0.220782 0.465012 0.611135

0010011 1000011 1110100 1101011 0001011 0000010 0000101 1111101 1011111 1011111   1.8499e+03   79 59934
-0.346972 0.611987 0.240613 0.130945 -0.502982 -0.601024 -0.578895 0.225829 0.429217 0.429449

0000001 1011111 1000010 1010111 0001111 0000110 0000101 1111101 1000010 1001111   1.8498e+03   69 52500
-0.620221 0.424206 0.601208 0.371894 -0.530317 -0.594846 -0.578596 0.220831 0.607811 0.535429

0011011 0000111 1011110 1000110 0001111 0000010 0000101 1111101 1001011 1001011   1.8495e+03   76 57346
-0.450645 -0.582436 0.435769 0.598795 -0.536029 -0.60305 -0.577441 0.221783 0.500674 0.505653

0010011 0001111 1011110 1000110 0001111 0000010 0000101 1111101 1011111 1001101   1.8495e+03   74 55562
-0.344686 -0.535807 0.436718 0.593605 -0.538529 -0.606781 -0.571912 0.220619 0.423833 0.546988
```

Figure 3-4: Example SAMPLE.MIN File

As pictured in the example, the first row of each entry lists a gray coded binary string, the fitness value of the string in man-days, the generation number, and the specific trial number in which the string was created. The second row contains the decimal values of the better strings prior to being processed and sent to the TEST.DNX file, the input file to the DYNAMICA program as described in the system overview on pages 29-30.

Before being sent to the TEST.DNX file, the fitness function must transform the coded WCWF-1 variables into valid parameters for the DYNAMICA program. This means they must be between 0 and 1 inclusive. When using the gray coding scheme, the GAUCSD creates 128 possible values ranging from -0.64 to +0.63 for each of the ten variables. The fitness function takes these values and adds 0.64 to create a range from 0.0 to 1.27, the smallest range required to cover all of the possible WCWF-1 values (recall that if the value is greater than 1, the value is changed to equal 1). These values can now be sent to the TEST.DNX file as input for DYNAMICA.

The other files pictured in Fig. 3-3 are the SAMPLE.OUT, the SAMPLE.CPT, and the SAMPLE.LOG files. The SAMPLE.OUT file is a listing of the minimum results at either the end of each generation or the "Report Interval" as assigned in the SAMPLE.IN file, which ever comes first. The important columns in this report are the best of generation and the generation average fitness. Both of these columns are pictured in the partial SAMPLE.OUT file shown in Fig. 3-5. The SAMPLE.CPT file is used by the GAUCSD as a checkpoint file. It contains all of the binary strings that were produced during the last full generation, and it allows for a restart of the GAUCSD with only the loss of the current generation's computations if the program is interrupted for any reason. The last file is the SAMPLE.LOG. This file records all of the historical data about when the GA run began, if and when it was restarted, and when and why it terminated.

```
                                                   Best of Gen    Avg. of Gen
 0   1000 0  0 -2.043  1.98566e+03  1.87226e+03  1.864130e+03  1.985661e+03
 1   2000 0  0 -2.085  1.96803e+03  1.86802e+03  1.863620e+03  1.950406e+03
 2   3000 0  0 -2.123  1.95660e+03  1.86653e+03  1.863520e+03  1.933721e+03
 3   4000 0  0 -2.160  1.94809e+03  1.86578e+03  1.863520e+03  1.922557e+03
 4   5000 0  0 -2.217  1.93755e+03  1.86436e+03  1.858520e+03  1.916929e+03
 5   6000 0  0 -2.233  1.93353e+03  1.86352e+03  1.858520e+03  1.909378e+03
 6   7000 0  0 -2.252  1.93032e+03  1.86290e+03  1.858520e+03  1.907883e+03
 7   8000 0  0 -2.295  1.92467e+03  1.86189e+03  1.857240e+03  1.901473e+03
 8   9000 0  0 -2.309  1.92214e+03  1.86147e+03  1.857240e+03  1.896901e+03
 9  10000 0  1 -2.323  1.91981e+03  1.86111e+03  1.857140e+03  1.894122e+03
10  11000 0  1 -2.357  1.91576e+03  1.86053e+03  1.856970e+03  1.890580e+03
```

Figure 3-5: Example SAMPLE.OUT File

To display the simulation results, a graphical format has been chosen. The decimal values from each entry in the SAMPLE.MIN file will be entered into a spreadsheet to allow for easy manipulation. After adding 0.64 to each of the values and ensuring none of the values are greater than 1 (as explained on page 35), it will then be possible to display the results by utilizing the spreadsheet's graphing function. Other data that will be used to display simulation results is a graph of the Total Work Force (TOTWF) as determined by the DYNAMICA program. This graph depicts the total work force levels that were used throughout the development process. Graphically viewing the WCWF policies and the TOTWF levels in this manner will make it easier to compare the results from each of the saved runs. Figures 3-6 and 3-7 illustrate graphical examples of a WCWF policy and the project's TOTWF levels respectively.

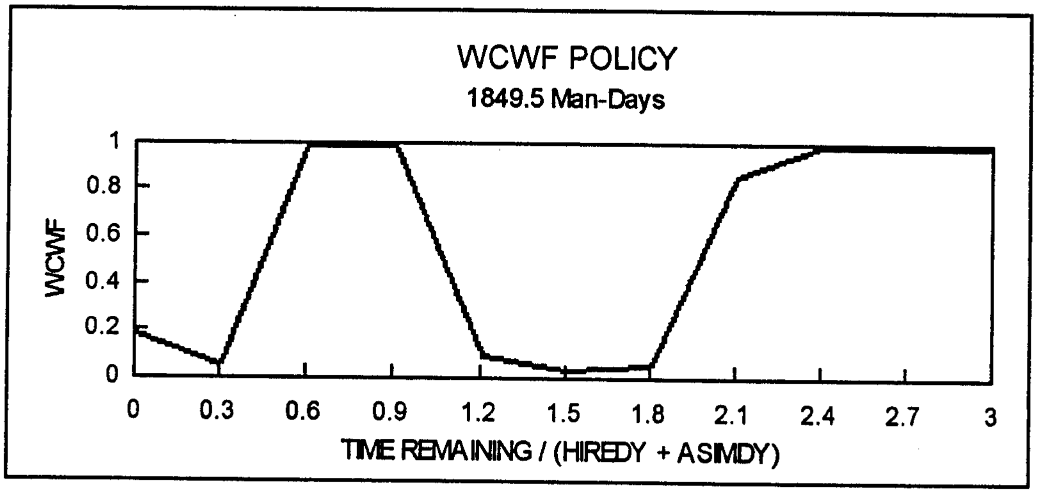


Figure 3-6: Sample WCWF Policy

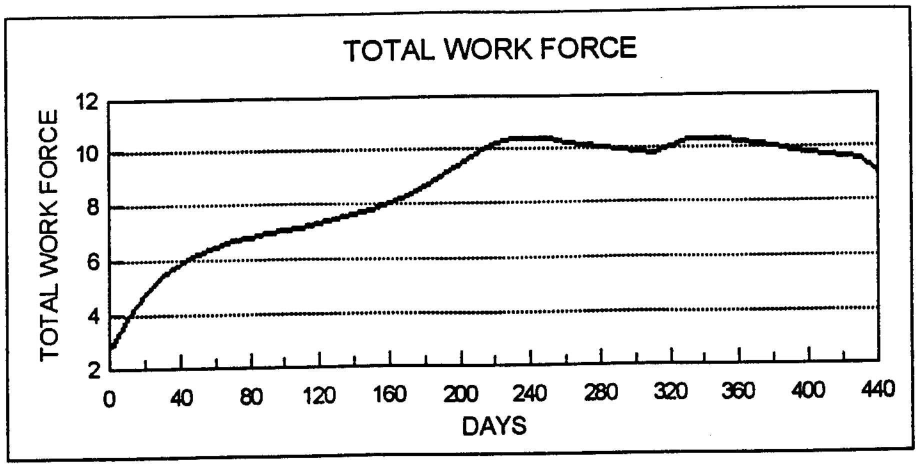


Figure 3-7: Sample Project TOTWF Levels

C. EXECUTING THE DYNAMICA PROGRAM

As discussed earlier on page 28, the DYNAMICA SD (DYNAMICA Model of Software Development) is a PC based program designed to simulate software project models. The simulation program is capable of running in two modes: (1) interactive and (2) batch. In the interactive mode, the user must negotiate numerous menus, and make several manual entries to choose between possible simulation models, make changes to the data, view or print the results, or do any of the other numerous alternatives. This mode of operation works fine when there are only a couple of simulations to be run, but it becomes impractical when tens of thousands of simulation runs, each with differing inputs, are required.

When coupled with genetic algorithms, simulation runs into the thousands are likely. In this case, the batch mode of operation is most effective. To operate the DYNAMICA program in the batch mode, it is necessary to enter a series of DOS commands. These commands are shown in Fig. 3-8.

```
DYNEX EXAMPLE1 TEST -OUT TEST.DTM -D TEST.DRS

SMLT EXAMPLE1 -GO TEST.RSL -DTM TEST

REP TEST.RSL REPORT.DRS -T
```

Figure 3-8: DYNAMICA DOS Batch Mode Execution Commands

The first line of the commands listed in Fig. 3-8 invokes the DYNAMICA program. It specifies which mode to use (*DYNEX*), which example to simulate (*EXAMPLE1*), and the name of the input file (*TEST*, the .DNX is assumed). It also includes an *-OUT* switch which specifies the file containing the output (*TEST.DTM*) and a *-D* switch which redirects the text output to the *TEST.DRS* file. The second of the command lines executes the simulation procedures (*SMLT*) and specifies which data to use (*EXAMPLE1*). This command also executes the simulation without requiring user confirmation through the *-GO* switch, specifies the name of the results file (*TEST.RSL*), and designates the .DTM file to be used in the simulation (*TEST.DTM*) with the *-DTM* switch. The last of the three command lines is used to execute the report procedures (*REP*). This command uses the file (*TEST.RSL*) generated earlier to produce a report whose format is specified in the *REPORT.DRS* file. This report, given the name TEST.OUT in the *-T* switch, is later used in the determination of the GAs' fitness value.

D. ROLE OF THE FITNESS FUNCTION

The heart of the GASD model is the fitness function. While the fitness function has been addressed as a separate system component, it is actually an embedded part of the

GAUCSD. The fitness function, written in the C programming language, is compiled as part of the genetic algorithm. An example of the fitness function used in this research project can be found in the Appendix.

It is the fitness function that initiates the entire simulation procedure. When executing the GASD, the fitness function creates a file (TEST.DNX as shown in Fig. 3-9) containing the WCWF-1 variables. The fitness function then executes the DYNAMICA program using the batch mode commands pictured in Fig. 3-8. Following execution, the DYNAMICA program writes a TEST.OUT file and passes it back to the fitness function which extracts the Total Man-Days value and sends it to the GA to start the process over.

```
T TWCWF1=0.966554 1.000000 1.000000 0.119963 0.097261
              0.041829 0.172365 0.830645 1.000000 1.000000 1
C ADMPPS=0.5
C DSIPTK=40
C MXSCDX=1E6
C TRPNHR=0.25
C TMPRMR=50
C UNDEST=0.35
C ASIMDY=20
C DEVPRT=0.85
C HIREDY=30
C RJBDSI=24400
C TOTMD1=1111
C AVEMPT=1000
C INUDST=0.4
C TDEV1=320
T TNERPK=24 22.9 20.75 15.25 13.1 12
T TPFMQA=0.325 0.29 0.275 0.255 0.25 0.275 0.325 0.375 0.4 0.4 0
```

Figure 3-9: Example TEST.DNX File

E. GASD MODEL EXECUTION

1. Hardware and Software Requirements

The primary hardware platform that was used in this research was a 66 MHz 486DX Personal Computer (PC). Although it is possible to run the GASD simulation program on less capable machines, the high speed capabilities of the 486DX-66 processor allowed for much shorter simulation runs. Another hardware platform that was utilized to run the GASD program were three 386 CPU personal computers hooked up to a Local Area Network (LAN). This LAN configuration was great for running simultaneous simulations, but the processing performance/speed was severely degraded.

Most of the software requirements for the project have already been discussed in detail. These requirements include the GAUCSD 1.4, including the fitness function, and the DYNAMICA program. The final software requirement for this system is a C compiler. The compiler version selected for this project was the Borland C++ Ver. 3.1. This version is compatible to both the DOS and Windows environments, and it is able to produce DOS executable programs, a necessity to execute the commands listed in Fig. 3-8.

2. Compiling the GASD Program

To execute the GASD program, all of the system components must be compiled and linked together. To do this, the GAUCSD-1.4 files and utilities must be ported to the DOS environment. The recommended procedure to do this is thoroughly explained in both Ref. 9:pp. 4-5 and Ref. 6:p. 43-45. All of the defined files are maintained in four directories (ETC, GAWK, SRC, and USR) as discussed in [Ref. 6:p. 44].

The SRC directory contains all of the GAUCSD source files and all of the DYNAMICA program files. The USR directory contains the formatted fitness function (GOLD_GA.C), which is used by the GAUCSD, and various other .C files. The ETC directory contains a wrapper function for embedding the fitness function in the GAUCSD, and the GAWK directory contains the GAWK utility. Since various versions of the GAWK utility that were available in the public domain were not compatible with this system, the addition of the wrapper around the fitness function was accomplished by editing a sample program file (the GOLD_GA.C [Ref. 6: Appendix] with wrapper already attached). The edited GOLD_GA.C (located in the appendix of this thesis) was then ready to be compiled with the rest of the program. The next step was to setup a project file in the C++ program.

The first step in creating a project file is to open the C++ application. When the program is open, select the *Project* pull-down menu and select *New project...*. In the window that appears, enter the desired project path and name (the example in Fig. 3-10 used c:\src\gaucsd.ide). Other items that need to be selected in this window are the target type (application.exe) and the platform (DOS Standard). Once these choices have been made select *OK*.

The next step is to modify the project window that appears (if it does not appear select the *View* pull-down menu and select project). Inside the project window highlight the *filename*[.cpp] line and delete it. The next step is to add the required files to the project. This is done by choosing the *Add* item to project selection. This will bring up the

Add to Project List window. Change the directory in the window to the USR directory, highlight the gold_ga.c file, and select *Add*. Then change the directory to SRC and add all of the remaining files listed in Fig. 3-10. Ensure that the gold_ga.c file is the first file added to the project list. If not, the program will not compile.

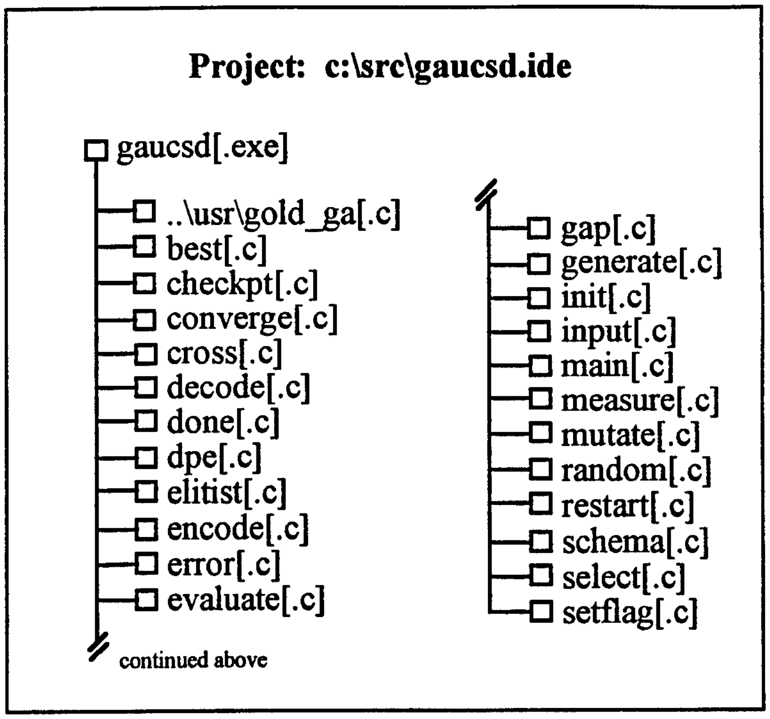


Figure 3-10: GASD Project File

When all of the files in Fig. 3-10 have been added to the project list, it is time to compile the program. This step is done by selecting the *Project* pull-down menu and selecting *Make all*. If things go right, the program will compile successfully and add an executable file to the directory that was specified by the path selection earlier. For the example in Fig. 3-10, the executable file GAUCSD.EXE would be created in the SRC directory.

3. Executing the GASD Program

Once the GASD program has been successfully compiled, it is ready for execution. The command to execute the example program is:

GAUCSD SAMPLE

where GAUCSD is the executable file that was created during the compiling process and SAMPLE is the initialization file for the simulation run (see Fig. 3-2 page 32). During the simulation runs, the GA will create the SAMPLE.MIN, SAMPLE.OUT, SAMPLE.CPT, and SAMPLE.LOG files in the same directory that the SAMPLE.IN file is located.

IV. EXPERIMENTATION AND RESULTS

A. INITIAL TESTING

Initial testing of the modified GASD model showed that the model was functioning correctly and attempting to zero in on a Willingness to Change Work Force (WCWF) Policy that would produce an optimal solution. Initial tests were conducted with a population size of 1000 and set to run for 100 generations. After several runs terminated after only one generation (due to a memory problem), the population size was changed to 900. With this minor change, the GASD model was able to execute all 100 generations. It should be noted that numerous attempts were made to correct the memory problem, but after many hours of editing, compiling, recompiling, and executing the model, a solution to solving the memory allocation problem could not be found.

During the GASD model execution, the 486DX-66 platform was able to perform approximately 1450 trials per hour which resulted in a new generation approximately every 40 minutes. At this rate, the GASD required 67 hours or almost 3 days of processing time to run all 100 generations.

The initial simulation runs were also used to see if varying the crossover rate had any effect on the model's performance. The recommended crossover rate is between 60 and 70% [Ref. 6:p. 47], therefore, the values that were tested were 0.5, 0.7, and 0.8 (0.6 was not tested during these trials because data was available from earlier runs done at this rate). After almost six days of continuous processing, the results of the three runs and previous results utilizing the 0.6 crossover rate were compared. Examination of the results failed to show any conclusive evidence that one crossover rate was better than the others. Therefore, all remaining tests were run using the 0.6 value.

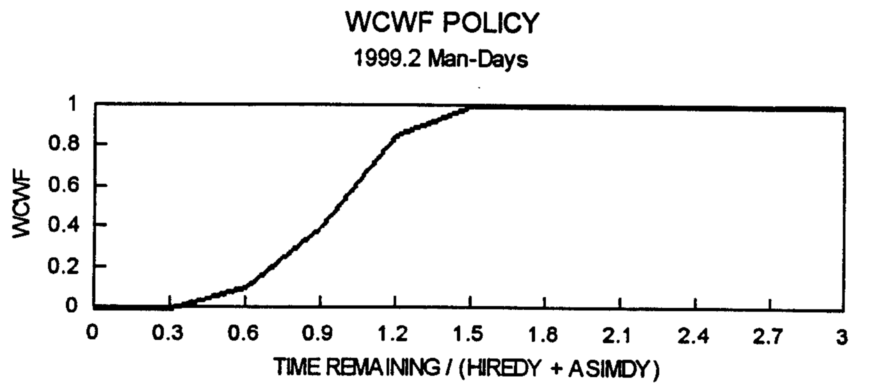
B. FINAL TESTING

The final testing was done in three distinct phases. In Phase One, the Willingness to Change Work Force (WCWF) variable was made solely a function of WCWF-1. Therefore, the WCWF-2 variable was ignored when calculating the Total Man-Days value. This was done by changing the Maximum Schedule Completion Data (MXSCDX) value located in the fitness function to 1E6 or 1,000,000 (see Appendix).

In Phase Two of the testing, the MXSCDX value was changed back to its original value of 1.16. This meant that the WCWF-2 variables would come into play when nearing the Maximum Tolerable Completion Date (Maximum Schedule Completion Data x Time to Develop).

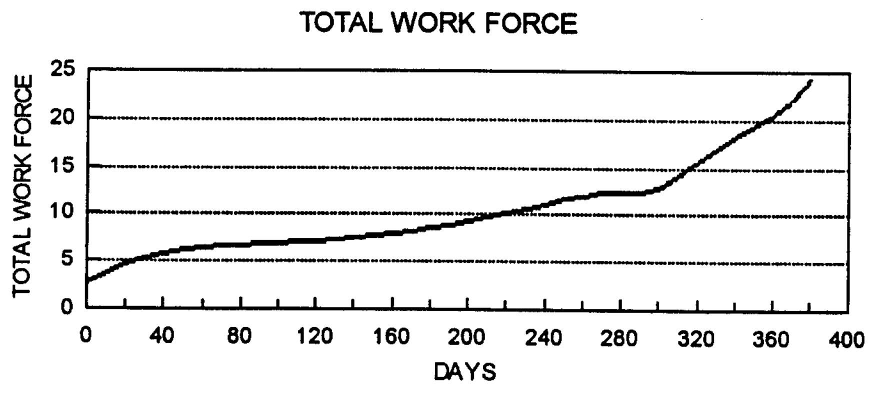
In the final phase of testing, Phase Three, the MXSCDX variable was again changed to 1E6 in the attempt to base the WCWF variable solely on WCWF-1. The variation in this phase was that the optimal Planned Fraction of Manpower Quality Assurance (TPFMQA) values found to minimize the Total Man-Days during earlier research [Ref. 6:p. 57] were used in place of the values found in the fitness function (see Appendix). The TPFMQA values that were used were 0.2546, 0.1684, 0.1319, 0.1083, 0.1035, 0.1123, 0.1036, 0.1120, 0.1167, 0.1035, and 0.

Prior to discussing any findings, a reference value or control must be established. This was done by manually running the GASD model with the original values. The results from the simulation can be found in Fig. 4-1. Fig. 4-1 (a) is a graph of the original WCWF-1 values. Fig. 4-1 (b) is the graph of the Total Work Force (TOTWF) levels determined during the simulation.



0	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4	2.7	3.0
0	0	0.1	0.4	0.85	1	1	1	1	1	1

(a)



(b)

Figure 4-1: Control Results

1. Phase One - WCWF-1 Policy

The first step in Phase One was to run a manual simulation and change only the MXSCDX value to 1E6. Since all other input values remained the same as those used in the control simulation, the WCWF-1 graph was identical to the graph pictured in Fig. 4-1 (a). The Total Work Force graph, however, changed as seen in Fig. 4-2.

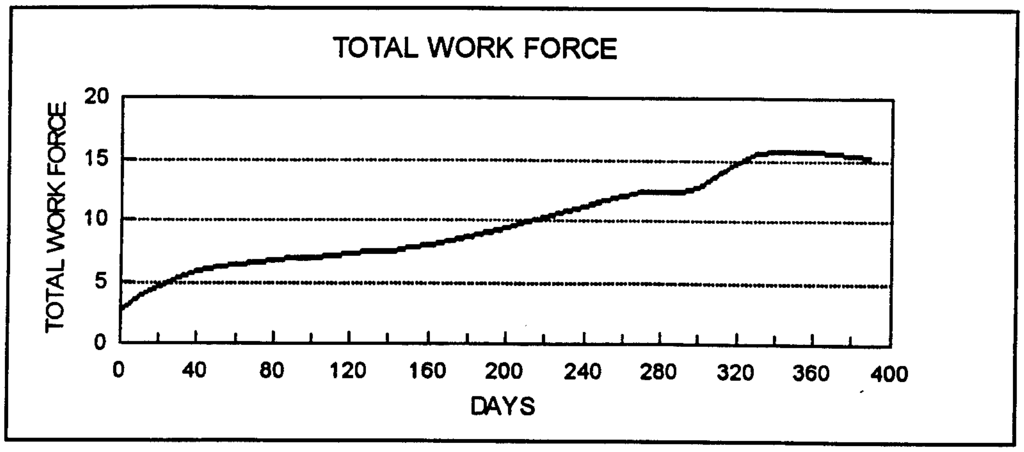
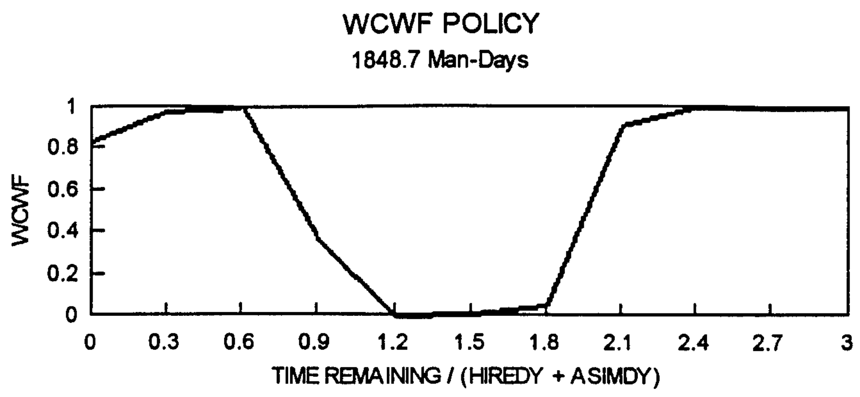


Figure 4-2: TOTWF (Where MXSCDX = 1E6)

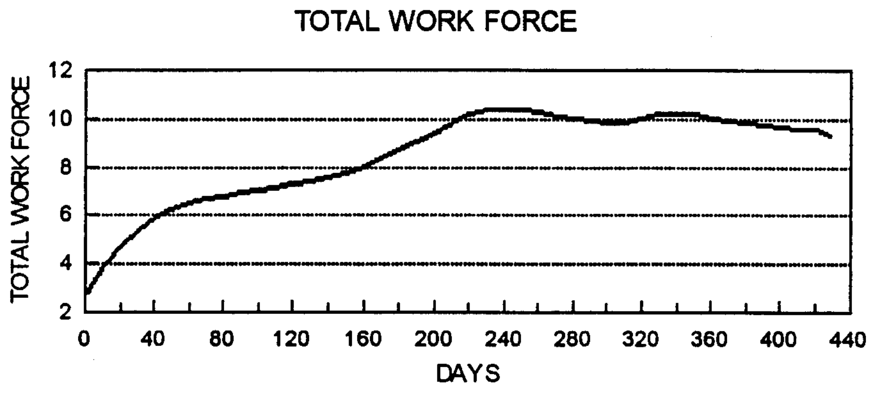
This modified simulation required only 1953.7 Man-Days, that is 45.5 Man-Days less than the control's 1999.2 requirement. The major difference between the two Total Work Force graphs is that the control's total work force continues to climb steadily at the 330 day point while the graph pictured in Fig. 4-2 levels off at the same point. A close examination of the two graphs also shows that the control simulation was completed in approximately 380 calendar workdays while it took approximately 10 more workdays to complete the simulation pictured in Fig. 4-2.

The second step in this phase was to run the GASD model with the MXSCDX variable equal to 1E6. Several simulations were run in this phase and the WCWF-1 policy curve that best minimized the Total Man-Days and the corresponding TOTWF graph are illustrated in Fig. 4-3 (a) and (b) respectively.



0	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4	2.7	3.0
0.830	0.945	1	0.368	0.002	0.012	0.052	0.9	1	1	1

(a)



(b)

Figure 4-3: GASD Results - Phase One

When comparing the results from part one and part two of this phase, there is a major difference in the Total Man-Days values. The GASD simulation required only 1848.7 Man-Days to complete the project while the manual simulation (using the original

WCWF-1 values) required 1953.7 Man-Days, a difference of 105 Man-Days. One draw back to the GASD solution is that it took approximately 40 more calendar workdays to complete than the manual simulation. A close examination of the TOTWF plots shows that the GASD's work force leveled off at day 210, while the manual simulation's work force grew until day 320.

2. Phase Two - Utilizing WCWF-1 and WCWF-2 Policies

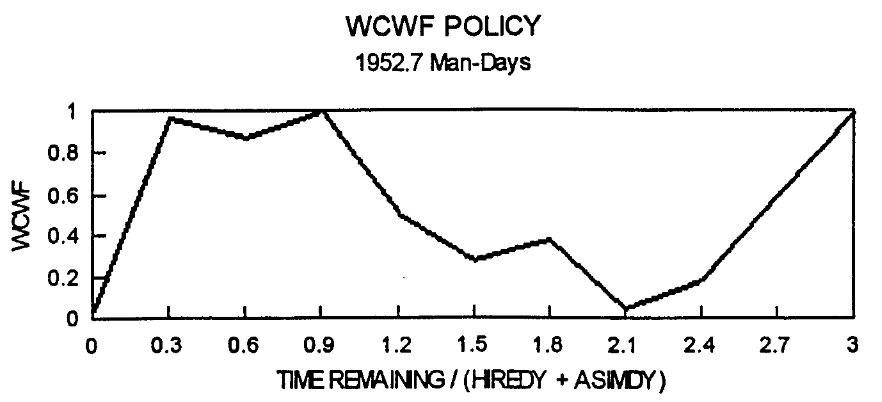
The second phase of this project involved running the GASD model with the MXSCDX variable equal to 1.16. Again, several simulations were run in this phase and the WCWF-1 policy curve that best minimized the Total Man-Days and the corresponding TOTWF graph are illustrated in Fig. 4-4 (a) and (b) respectively.

In this phase of the experimentation, the WCWF-2 data came into play. As stated in Chapter II;

$$WCWF = MAXIMUM\,(WCWF\text{-}1,\ WCWF\text{-}2).$$

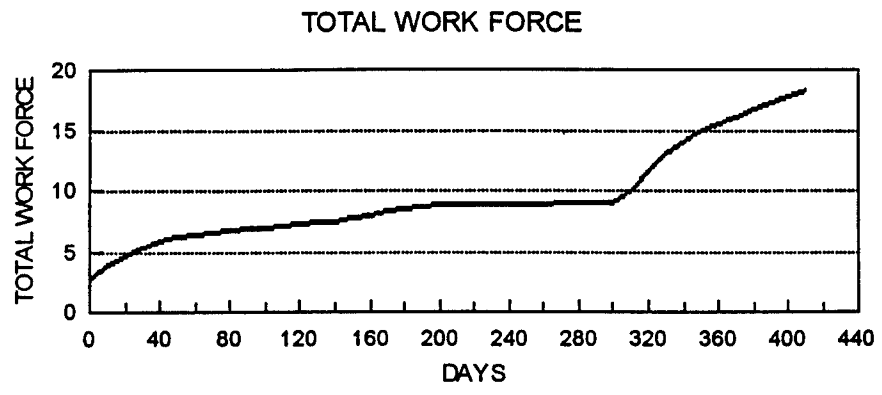
As the "Scheduled Completion Date" approaches the "Maximum Tolerable Completion Date," schedule pressures start to develop causing WCWF-2 to gradually rise and eventually exceed WCWF-1, when this happens, the "Willingness to Change Work Force" function becomes based solely on WCWF-2. In this research, the original WCWF-2 variables were used during the GASD simulations.

In this phase the GASD simulation required only 1952.7 Man-Days to complete the project, a 46.5 Man-Day improvement over the 1999.2 Man-Days required by the control simulation. This improvement did not come without a penalty though, the GASD simulation required approximately 30 extra calendar workdays to complete the project.



0	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4	2.7	3.0
0.038	0.968	0.870	1	0.506	0.277	0.38	0.048	0.184	0.59	1

(a)



(b)

Figure 4-4: GASD Results - Phase Two

3. Phase Three - Utilizing Optimized TPFMQA Values

The first step in Phase One was to run a manual simulation with a MXSCDX value of 1E6 and the TPFMQA values of 0.2546, 0.1684, 0.1319, 0.1083, 0.1035, 0.1123, 0.1036, 0.1120, 0.1167, 0.1035, and 0. In this simulation, the WCWF-1 input variables were the same as those listed in Fig. 4-1 (a). TOTWF results from this manual simulation are shown in Fig. 4-5.

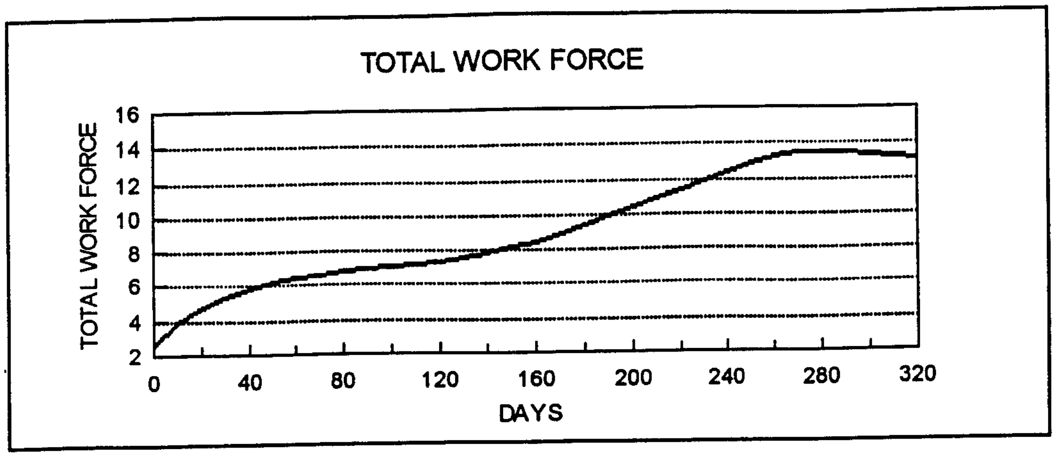
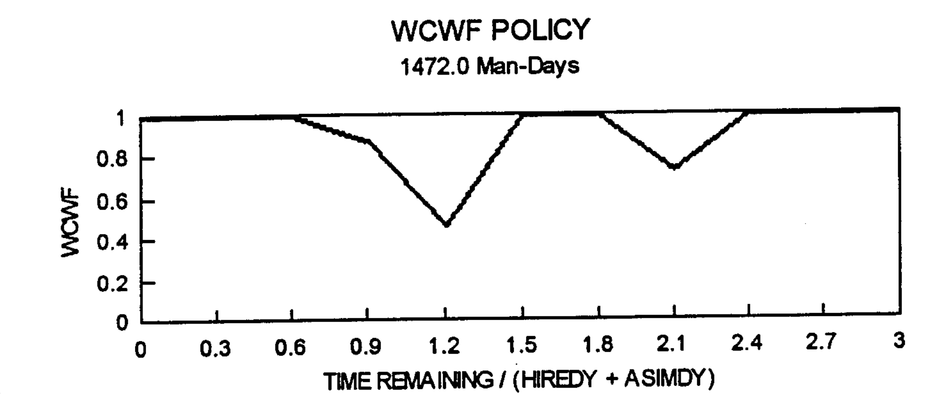


Figure 4-5: TOTWF (Modified TPFMQA Variables)

The next step in this phase was to modify the fitness function to include the optimal TPFMQA variables. Once this was done and the fitness function was recompiled, a GASD simulation was run. The WCWF-1 policy curve that best minimized the Total Man-Days and the corresponding TOTWF graph from the GASD simulation are show in Fig. 4-6 and Fig. 4-7 respectively.



0	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4	2.7	3.0
1	1	1	0.875	0.462	1	1	0.73	1	1	1

Figure 4-6: GASD WCWF-1 Policy Curve - Phase Three

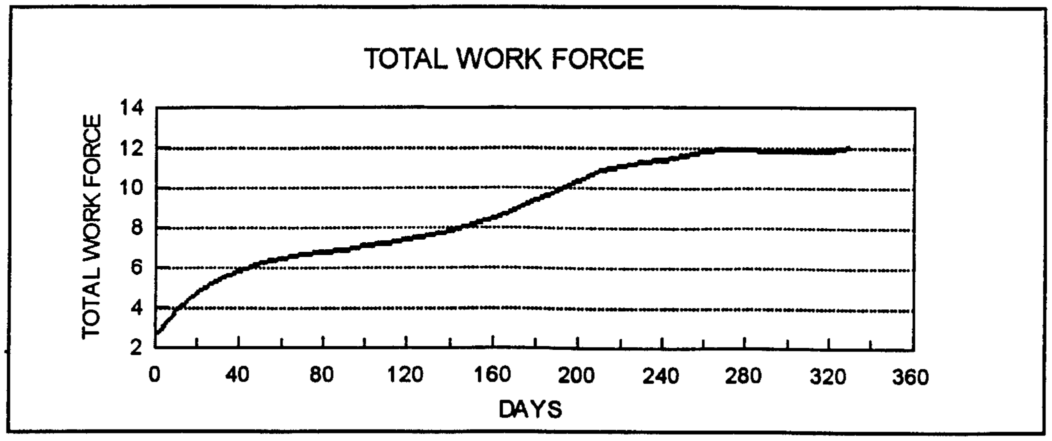


Figure 4-7: GASD TOTWF Results - Phase Three

In the final phase, the GASD simulation required 1472.7 Man-Days to complete the project, while the manual simulation required 1475.5 Man-Days. The GASD model was again able to lower the Man-Day requirement, but in this case it was only by 3.5 Man-Days. To find this slight reduction in Man-Days, the GASD model required approximately 10 more calendar workdays to complete the simulation.

C. FINAL OBSERVATIONS

When reviewing all of the results, there are several observations that can be made. First, in each of the three phases the GASD model provided a solution that required less Man-Days to complete. In Phase One, the GASD model required only 1848.7 Man-Days while the manual simulation required 1953.7, a savings of 105 Man-Days. In Phase Two, the GASD model required 1952.7 Man-Days to complete the project, 46.5 Man-Days less than the 1999.2 required by the control simulation. In Phase Three, even though the difference was only 3.5 Man-Days, the GASD model was still able to simulate a more effective WCWF-1 policy that would help minimize development costs.

A second observation that can be made when reviewing all of the results is that every time the GASD model minimized the Total Man-Day requirements the workdays required to complete the project increased. During Phase One, the GASD simulation required approximately 40 additional calendar workdays to complete the project. During Phase Two and Three, GASD added approximately 30 and 10 workdays respectively.

Upon closer review, another observation that can be made is that in each phase of the experimentation the simulation that maintained the fewest personnel and leveled-off the soonest had the lowest Man-Day requirements. This implies by hiring less personnel, the Man-Day count remained lower, but due to having fewer personnel, it took more workdays to complete the projects.

The above results are a contradiction to Brook's Law. Brook's Law states that adding manpower to a late software project makes it later [Ref. 5:p. 109]. In the three phases, if Brook's Law were applicable, the simulations that added several more workers at the end of the late project should have taken longer than the simulations where the work force built to a point and then remained relatively constant. Instead, it appears that the opposite is true. This finding coincides with previous research where the authors found that adding more people to a late software project will always cause the project to become more costly, but not necessarily cause the project to be completed later [Ref. 5:p. 117].

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The focus of this research was to see if the GASD model [Ref. 6] could be modified enabling it to produce an optimal software project staffing policy that would minimize total project costs. This was accomplished by modifying the original fitness function [Ref. 6: Appendix] to process the WCWF-1 variables instead of the TPFMQA variables used in the DYNAMICA program. The resulting fitness function can be found in the appendix.

Experiments in this research were broken into three phases. Phase One focused on making the Willingness to Change Work Force (WCWF) variable solely a function of WCWF-1. This was done by changing the Maximum Schedule Completion Data (MXSCDX) value located in the fitness function to 1E6. The results of these tests showed that the GASD was able to develop a staffing policy curve that lowered the project cost by as much as 105 Man-Days.

In Phase Two of the testing, the original WCWF-2 variables were used in the GASD simulation. This was accomplished by changing the MXSCDX value back to the original value of 1.16. The results of these tests showed that the GASD was again able to develop a staffing policy curve that lowered the project cost. This time, there was a 46.5 Man-Day improvement.

In Phase Three, the optimal Planned Fraction of Manpower Quality Assurance (TPFMQA) values found to minimize the Total Man-Days during earlier research were used. These values replaced the TPFMQA values found in the fitness function. In this phase, the MXSCDX variable was again changed to 1E6 to base the WCWF variable solely on WCWF-1. In this case, the GASD was able to develop an optimal staffing policy curve, but it lowered the project cost by only 3.5 Man-Days.

Although the GASD was able to minimize the total cost in each of the final three phases, there was a trade-off that had to be made. When the GASD model minimized the Total Man-Day requirements, the workdays required to complete the projects increased. Phase One of the experimentation required approximately 40 additional workdays to complete the project. Phase Two and Three added approximately 30 and 10 workdays respectively.

It was also noted that in each phase of the experimentation the simulation that maintained the fewest personnel and leveled-off the soonest had the lowest Man-Day requirements. These results indicate that by hiring less personnel, a manager could control the Man-Day expenditure, but due to having fewer personnel, the workday requirement to complete the project would increase.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

1. Continue Current Research

There are two recommendations for the continuation of the current research. The first recommendation is to develop an "implementable" staffing policy. This could be done by adding constraints (on the shape of the curves) to control the rate of change to the Total Work Force levels prior to running the GASD model optimization. The other recommendation is to define a more sophisticated objective function that would optimize both project cost and schedule over-run. The new objective function would allow the GASD to find an optimal solution that would appropriately weigh the project cost and any schedule over-run.

2. Increase the Number of Variables

Up until this point all of the testing using the GASD has involved only one variable at a time. The selection of these variables was based on the desire to study either the significance of the QA effort in software development [Ref. 6:p 70] or the significance of the staffing policy during software development. However, these variables are not the only parameters that influence the cost of development. Total project cost could be optimized based on a combination of given variables.

One example of this combination of variables would be to simultaneously process the TPFMQA and WCWF-1 variables. Results from the tests using the optimized TPFMQA variables while optimizing the WCWF-1 variable suggest that if the variables can be combined, the results may improve. Making these changes to the GASD model would not be difficult. Required changes to the model are discussed in Chapter III. Modifying the fitness function to include multiple variables would enable the GASD model to simultaneously process any number of given variables.

APPENDIX

/* This code, written in C is called by the GAUCSD genetic algorithm. It takes a bit string containing 10 double precision floating point numbers representing the input parameters from the genetic algorithm and converts the string into a double precision floating point representing the output value. To accomplish this, it calls DYNAMICA, a DOS based simulation program. Then, during its execution, the DYNAMICA program generates a report file from which this code extracts relevant data. This data is then used by the genetic algorithm to perform the fitness measure portion of the execution. */

```
#include <math.h>
#include <stdio.h>
double gold(x)  /* Designates function gold as a double precision float with x as the input
                   parameter. */

/* GAUCSD sends array x to this code. */

        register double *x;  {
        double p1,p2,p3,p4,p5,p6,p7,p8,p9,p10;  /* Space allocation for function gold. */
        char dea_file_name[80];  /* Space allocation for the DE-A file. */
        char rpt_file_name[80];  /* Space allocation for Report out_file_name. */
        char file_line[80];  /* Space allocation for the string file_name. */
        char command_line[80];  /* Space allocation for the string command_line. */
        char *c;  /* Space allocation for character pointer 'c'. */
        double result,mult;  */ Makes result double precision and multiplies result. */
        FILE *f;

* /This portion of the code takes the numbers from the array x and transforms them into
valid parameters for DYNAMICA. It then places the values into fields of p1 thru p10. */
        p1 = x[0] + .64 ; p2 = x[1] + .64 ; p3 = x[2] + .64 ;
        p4 = x[3] + .64 ; p5 = x[4] + .64 ; p6 = x[5] + .64 ;
        p7 = x[6] + .64 ; p8 = x[7] + .64 ; p9 = x[8] + .64 ; p10 = x[9] + .64 ;
```

```
* /This portion of the code checks the numbers in fields p1 thru p10 to see if they are
greater than 1.0.  If they are, they program converts them into 1.0. */

        if(p1>1.0) { p1=1.0;}
        if(p2>1.0) { p2=1.0;}
        if(p3>1.0) { p3=1.0;}
        if(p4>1.0) { p4=1.0;}
        if(p5>1.0) { p5=1.0;}
        if(p6>1.0) { p6=1.0;}
        if(p7>1.0) { p7=1.0;}
        if(p8>1.0) { p8=1.0;}
        if(p9>1.0) { p9=1.0;}
        if(p10>1.0) { p10=1.0;}

        sprintf(dea_file_name,"TEST.dnx");  /* Names the file with the .DNX extension
                necessary for DYNAMICA to input the file. */

        f = fopen(dea_file_name,"w");  /* Opens the file for writing. */

        if(!f) { perror("gold"); exit(1); }  /* Error condition to ensure file is opened.
                If the file is not opened, the program exits the simulation. */


/* The following information is written directly into the TEST.DNX file.  This data is
determined by the genetic algorithm and changes each time the file is created. */

        fprintf(f,"\n\

T TWCWF1=");

        fprintf(f,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf 1\n",
                p1,p2,p3,p4,p5,p6,p7,p8,p9,p10);


/* The following information is written directly into the test.DNX file.  This data remains
constant each time the file is created. */

        fprintf(f,"\
C ADMPPS=0.5\n\
C DSIPTK=40\n\
C MXSCDX=1E6\n\
C TRPNHR=0.25\n\
C TMPRMR=50\n\
C UNDEST=0.35\n\
C ASIMDY=20\n\
C DEVPRT=0.85\n\
C HIREDY=30\n\
C RJBDSI=24400\n\
C TOTMD1=1111\n\
C AVEMPT=1000\n\
C INUDST=0.4\n\
C TDEV1=320\n\
T TNERPK=24 22.9 20.75 15.25 13.1 12\n\
T TPFMQA=0.325 0.29 0.275 0.255 0.25 0.275 0.325 0.375 0.4 0.4 0\n");
        fclose(f);  /* Closes the file. */

/* Prints the DYNEX command to a character array named command_line. */

        sprintf(command_line, "DYNEX EXAMPLE1 TEST -OUT TEST.DTM -D TEST.DRS");

/* Sends command_line to DOS which calls the DYNAMICA program for execution.
If an error code is detected, program is exited and an error message is sent. */

        if(system(command_line)) {

                perror("system:");

                exit(1);  }
```

```
/* Prints the SMLT (simulate) command to a character array named command_line. */

        sprintf(command_line,"SMLT EXAMPLE1 -GO TEST.RSL -DTM TEST");


/* Sends command_line to DOS.  If an error code is detected, program is exited and an
error message is sent. */

        if(system(command_line)) {

                perror("system");

                exit(1);  }


/* Prints the REP (report) command to a character array named command_line. */

        sprintf(command_line,"REP TEST.RSL REPORT.DRS -T");


/* Sends command_line to DOS.  If an error code is detected, program is exited and an
error message is sent. */

        if(system(command_line)) {

                perror("system");

                exit(1);  }

        sprintf(rpt_file_name,"TEST.out");  /* Assigns test file name TEST.out. */

        f = fopen(rpt_file_name,"r");  /* Opens TEST.out. */


/* Total man day data is retrieved from the file TEST.out and returned to the GA. /*

        while(fgets(file_line,79,f)) {

                if(!strncmp(file_line,"    TOTAL MAN-DAYS",18)) {

                        printf(" ---  %s",file_line); // compute result
```

/* Skips over file data until a number's reached. Once the correct number is found, that string of numbers is turned into a floating point number. */

```
            for(c = file_line; *c && !isdigit(*c); c++)  ;
            result = 0.0;
            /* 'c' presumably points to the integer. */
            for(;*c && (isdigit(*c) || *c==','); c++)  {
                    if(*c == ',')
                            continue;
                    result *= 10.0;
                    result += *c-'0';  }
            if(*c == '.') {
                    c++;
                    for(mult = 0.1; *c && isdigit(*c); c++) {
                            result += (*c-'0')*mult;
                            mult = mult/10.0;  }  }
```

/* Error condition to check if retrieved result is less than zero. If this occurs, an error condition exists and the program is exited. */

```
            if(result <= 0.0)  {
                    fprintf(stderr,"check this out!\n");
                    exit(1);  }
            fclose(f);  /* Closes the file. */
            fprintf(stderr,"result -- %lf\n",result);
            return(result);  }  }  /* Returns the number to GA for evaluation. */
```

/* This portion of the code is executed only if it was not possible to locate the total man days figure. If it gets here, the program will be terminated. */

```
    fclose(f);
```

```
        fprintf(stderr,"gold\n");
        exit(1);  }
```

/* GAeval gold 7:0.64dg10 */ //** AWK script necessary to process the gold.c program. This is a necessary directive for the post processor to ensure that this entire function is embedded correctly in the GAUCSD code. **// [Ref. 6:pp. 73-77]

LIST OF REFERENCES

1. United States General Accounting Office, *Mission-Critical Systems: Defense Attempting to Address Major Software Challenges*, Government Printing Office, Washington, DC, December 1992.

2. Subcommittee on Investigations and Oversight, *Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation*, Government Printing Office, Washington, DC, September 1989.

3. United States General Accounting Office, *Software Tools: Defense Is Not Ready to Implement I-CASE Departmentwide*, Government Printing Office, Washington, DC, June 1993.

4. United States General Accounting Office, *Embedded Computer Systems: Defense Does Not Know How Much It Spends on Software*, Government Printing Office, Washington, DC, July 1992.

5. Abdel-Hamid, Tarek K., "The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach," *IEEE Transactions on Software Engineering*, Volume 15, Number 2, February 1989.

6. Elliot, Donald M., *Application of a Genetic Algorithm to Optimize Quality Assurance in Software Development*, M.S. Thesis, Naval Postgraduate School, Monterey California, September 1993.

7. Burtka, Michael, "Genetic Algorithms," *The Stern Information Systems Review*, Volume 1, Number 1, Spring 1993.

8. Goldberg, D. A., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

9. Schraudolph, Nicol N., and Grefenstette, John J., *A User's Guide to GAucsd 1.4*, Computer Science & Education Department, UC San Diego, La Jolla, CA, July 1992.

10. Abdel-Hamid, Tarek K., and Madnick, Stuart E., *Software Project Dynamics an Integrated Approach*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990.